Tswap Protocol Audit Report

Version 1.0

www.github/kwamentw

December 22, 2023

Protocol Audit Report

Kwame 4B

March 7, 2023

Prepared by: Kwame 4B Lead Auditors: - Kwame 4B

Table of Contents

- Medium
  - [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline
  - [M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant
- Low
  - [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
  - [I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is not used the codebase and has to be deleted
  - [I-2] constructor lacks zero address check
  - [I-3] `Poolfactory::createPool` should use `.symbol()` instead of `.name()`
  - [I-4] 3 events should be indexed
  - [I-5] MINIMUM_WETH_LIQUIDITY should not be used in IF comparison
- Gas
  - [G-1] Unused line of code should be deleted

Protocol Summary

This protocol was basically created to swap tokens at a fair rate.

Disclaimer

Kwame 4B makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

../src/PoolFactory.sol ../src/TSwapPool.sol ## Roles - Users: Users use this protocol anytime theu want to swap their token - Liquidity provider: They provide the contract with liquidity and expand their pool

Executive Summary

- A lot of thought process went behind this audit, we did pay attention to details
- We spent almost a week on auditing this protocol
- we spent 3.5 hours reviewing the contract ## Issues found

Severity	Number os issues found
High	4
Medium	2
Low	2
Info	5
Total	13

Findings

High

[H-1] lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, the `swapExactOutput` function should specify a `maxInputAmount`.

Impact: If market conditions change before the transaction processes, the user could get a much worse swap.

Proof of concept: 1. The price of 1 WETH now is 1000 USDC 2. User inputs a `swapExactOutput` looking for 1 weth 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = random 3. The function does not offer a maxInput Amount 4. As the transaction is pending in the mempool, the market changes! ANd the price moves HUGE -> 1WETH is now 10,000 USDC. 10x more than the user expected 5. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1000 USDC

Recommended Mitigation: We should include the `maxInputAmount` variable

```
    function swapExactOutput(
        IERC20 inputToken
+        uint256 maxInputAmount
    .
    .
    .

        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+        if (inputAmount > maxInputAmount){            revert();
        }


        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes user of the contract to pay high fees

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of tokens of output. However, the function currently calculates the resulting amount. When calculating the fee, it scales the amount to 10000 instead os 1000

Impact: Protocol takes more fees than expected from users

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-         return
            ((inputReserves * outputAmount) * 10000) /
            ((outputReserves - outputAmount) * 997);

+       return
            ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);

    }
```

[H-3] `\TSwapPool::sellPoolTokens` exchanges input and output tokens causing users to receive the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates th swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` function is the one that should be called. Because users specify the exact amount of input tokens, not output.

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality

Proof of concept: 1. check the current rates 2. input your amount 3. calculate manually how much you are supposed to get per rates 4. run this contract in your test env and see how much you get in return

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter(ie `minWethToReceive` to be passed to `SwapExactInput`)

```
function sellPoolTokens(
        uint256 poolTokenAmount
+       uint256 minWethToReceive
    ) external returns (uint256 wethAmount) {
-        return swapExactOutput(                i_poolToken,i_wethToken
    ,poolTokenAmount,uint64(block.timestamp));
+        return swapExactInput(               i_poolToken,
    poolTokenAmount,i_wethToken, minWethToReceive, uint64(block.
    timestamp));
}
```

Additionally it will be wise to add a deadline

[H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x*y=k`

Description: The protocol follows a strict invariant of `x*y=k`, where: - `x`: The balance of the pool token - `y`; The balance of WETH - `k`: The constant product of two balances

the following block of code is responsible for the issue

```
        swap_count++;
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
        }
```

This means, that whenever the balances change in the protocol, the ratio between the two amounts should remain constant, hence the `k`. However, this is broken due to the extra incentive in the `swap` function. Meaning that over time the protocol funds will be drained.

Impact: A user could maliciously drain the protocol of funds by doing alot of swaps and collecting the extra incentive given out by the protocol.

Most simply put the protocols core invariant is broken

Proof of concept: 1. A user swaps 10 times, and collects the extra incentive of `1000000000000000000` tokens 2. That user continues to swap till all the protocol is drained.

Proof of Code

place the following into `TSwapPool.t.sol`

```
    function testInvariantBroken() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));

        uint256 outputWeth = 1e17;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));

        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);


        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);
        assertEq(actualDeltaY, expectedDeltaY);


    }
```

Recommended Mitigation: Remove the extra incentive mechanism. If you want to keep this in, we should account for thte change in the x*y=k protocol invariant. Or we should set aside tokens in the same way we do with fees.

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline

Description: the `deposit` function accepts a deadline parameter, which according to the documentation is the "the deadline for the transaction to be completed by", However, this parameter is never used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavourable. and can be susceptible to MEV attacks

Impact: Transactions could beb sent when market conditions are unfavoureble to deposit, even when adding a deadline parameter

Proof of concept: The deadline parameter is unused

Recommended Mitigation: My recommended fix;

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
```

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

Description: This function

```
        swap_count++;
        //fee-on-transfer
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
        }
```

breaks our invariant test because it breaks the logic of the contract after 10 trnascations

Impact: Funds drainage

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` value should go second.

Impact: Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation: the order of the emission has to be changed like this;

```
- emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+ emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

[L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given

Description: The `swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declres the named return value `output` it is never assigned a value, nor uses an explicit return statement.

Impact: The return value will always be 0, giving incorrect information to the caller

Recommended Mitigation: please remove & add the following statements respectfully

```
{
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
-           inputReserves,
-           outputReserves);

+         output = getOutputAmountBasedOnInput(inputAmount, inputReserves,outputReserves);

```

```
-         if (outputAmount < minOutputAmount) {revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
          }
+         if (output < minOutputAmount) {revert TSwapPool__OutputTooLow(output, minOutputAmount);
          }


-         _swap(inputToken, inputAmount, outputToken, outputAmount);
+         _swap(inputToken, inputAmount, outputToken, output);
      }
```

Informational

[I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is not used the codebase and has to be deleted

```
-     error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] constructor lacks zero address check

```
constructor(address wethToken) {
+    // add zero address check
        i_wethToken = wethToken;
    }
```

Another constructor

```
constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+        //implement a zero address check
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-3] `Poolfactory::createPool` should use `.symbol()` instead of `.name()`

```
-         string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).name());
```

```
+        string memory liquidityTokenSymbol = string.concat("ts",
    IERC20(tokenAddress).symbol());
```

[I-4] 3 events should be indexed

```
event Swap(
        address indexed swapper,
        IERC20 tokenIn,
        uint256 amountTokenIn,
        IERC20 tokenOut,
        uint256 amountTokenOut
    );
```

```
three events params must be indexed.
```

[I-5] MINIMUM_WETH_LIQUIDITY should not be used in IF comparison

Constants are not advisable to used as comparison variables

Gas

[G-1] Unused line of code should be deleted

```
-           uint256 poolTokenReserves = i_poolToken.balanceOf(address(
    this));
```